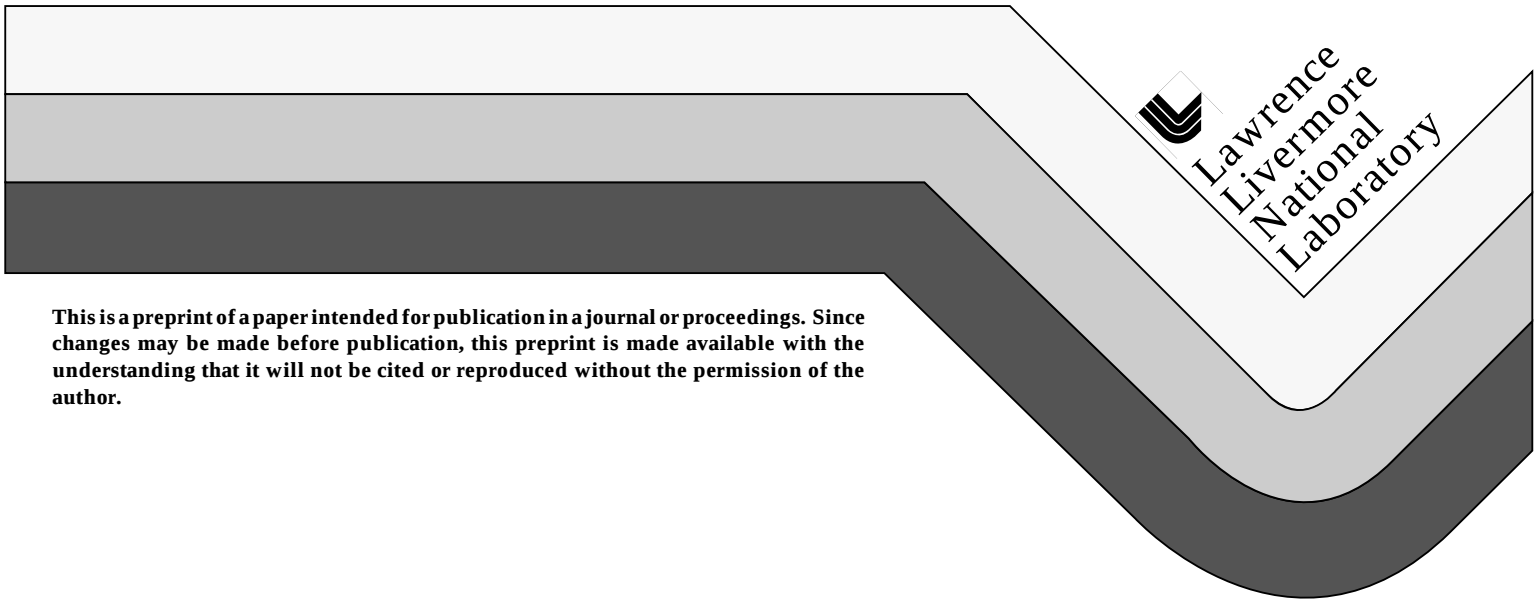
UCRL-JC-120461
PREPRINT

Security Profile Inspector (SPI)
The Next Generation

T. Bartoletti
J. Fisher

This paper was prepared for submittal to the
17th DOE Computer Security Group Training Conference
Milwaukee, WI
May 2-5, 1995

March 1995



Security Profile Inspector (SPI) *The Next Generation*

Tony Bartoletti[*] *(azb@llnl.gov)*
John Fisher (fisher23@llnl.gov)
Computer Security Technology Center
Lawrence Livermore National Laboratory

Abstract

The current Security Profile Inspector (SPI) conducts analysis of UNIX and VMS based operating system configurations to help system managers maintain secure operating environments. A broad horizontal range of security tests with a vertical array of usage options supports the needs of both novice and experienced system administrators. Its modular structure was designed to provide a foundation for distributed operation and network-wide inspection conducted from a central command host. The "Next Generation" SPI outlined here should serve to promote cost-effective computer security for many years.

Introduction

The relentless explosion in Internet growth has made computer and information security a crucial area of research. The transition from government and academically oriented Internet use to private and commercial use has lent a renewed sense of urgency to the search for solutions. The battle to safeguard information by eliminating computer system vulnerabilities

[*] Work performed under the auspices of the U. S. Department of Energy by the Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.
[*] Reference to any specific commercial products, process or service by trade name, trademark, manufacturer or otherwise, does not constitute or imply its endorsement, recommendation or favoring by the U.S. Department of Energy or the University of California.

and preventing intrusions continues unabated, and as new operating systems and new applications are adopted, new vulnerabilities arise to challenge the security technologist.

While most hackers still take advantage of fairly simple vulnerabilities, such as poorly chosen passwords or incorrect file permissions, an increasing number are employing automated "hacking tools," effectively allowing them to attack countless systems with a keystroke. These tools demonstrate an alarming degree of sophistication, exploit arcane and narrow system flaws, subvert key system software, and cover their tracks. It is of little consequence that the majority of casual hackers may not understand how these tools work. It is of great consequence that most system managers do not know how to protect themselves against such methods.

Proactive analysis of system security configurations is your first-line defense from intrusions. Following this front line are other aspects of computer security, including network monitoring and audit trail analysis. These are important once the intruder has already entered the system, either by having taken advantage of a vulnerability, or legitimately (disgruntled employee, etc.). But they are not a substitute for strong, proactive defense measures.

The approach of the Security Profile Inspector, or SPI, is prevention -- prevent the network-based hacker from entering the system in the first place, and prevent the "insider" (valid user or partially successful hacker) from escalating an attack to more sensitive parts of the system.

The SPI Project Objectives

The following points highlight the overall philosophy that has guided the SPI development effort:

Provide a comprehensive environment for security assessment

SPI provides a powerful set of tools for assessing computer protections. These tools analyze security configuration and access controls, identify outdated system software, and dynamically test for network service vulnerabilities. The results are presented in user-friendly reports, as well as in a machine readable format amenable to further data processing.

Design for upgrade ability to address new vulnerabilities

Computer security is a moving target. New software releases mean new vulnerabilities, new patches to be installed, and new configuration elements to monitor. In concert, a dedicated and well-connected community of Internet hackers continues to find new flaws in existing systems. For this reason, SPI was designed to be easily extensible, both by the developers and by the end user.

Address as wide a variety of platforms as possible

Every SPI revision has stressed increasing portability, providing consistent security management across a broad range of network-capable systems, because in a networked environment every system is a potentially weak link.

The Evolution of SPI

The November 1988 Morris or "Internet" Worm attack demonstrated just how vulnerable and interdependent network-based systems were. [1] Major Internet nodes were choked into failure, and in the resulting panic many subnets and systems shut themselves off from the greater Internet. Although this attack exploited vulnerabilities in a particular subset of UNIX systems, all Internet systems suffered days of service disruption and weeks of uncertainty as costly "clean-up" activities swallowed up otherwise productive time. Attacks such as the Morris Worm taught that *all* computers were vulnerable, particularly if they were part of the growing Internet community. Efforts to address computer system vulnerabilities were made on several fronts.

At Lawrence Livermore National Laboratory, a group was formed to provide coordinated computer incident response, as well as software tools, to help secure systems for the DOE. This organization, now known as the Computer Security Technology Center (CSTC) is comprised of the Computer Incident Advisory Capability (CIAC) and a number of research and development efforts in computer security.

The SPI effort was formed in response to the Internet Worm incident, in particular to help identify the security weaknesses in host systems and to aid in the detection and clean-up of

future incidents. With direct funding from the DOE Office of Security Affairs, a small initial set of system inspection tools was developed and deployed as the SPI 1.x series. This package included password testing, file content and attribute change detection, and a rudimentary user interface.

During this period, Dan Farmer at Purdue University put together a collection of many small UNIX "shell" scripts addressing scores of varied UNIX flaws, gathering many of these from system administrators across the Internet. The result was the now-familiar COPS suite of freeware UNIX security inspection tools. [2] The COPS suite was added to the SPI 2.x series, along with major improvements to the SPI user interface such as online, context sensitive help. (With SPI 3.x, COPS functionality was replaced by scripts written in SPI's Configuration Query Language.)

The transition of SPI from the 2.x to 3.x series involved considerable restructuring, both to help position SPI against obsolescence, and to satisfy some of the criticisms expressed by users. The isolation of the data extraction functionality into a layer separate from the data analysis routines was due in large part to suggestions by Russell Brand, a security consultant associated with Lawrence Livermore National Laboratory. This isolation allowed the majority of platform porting work to focus upon a limited set of code libraries.

The SPI developers have also received guidance from users in the field. A major contributor was Gene Rackow of Argonne National Laboratory. Gene remarked that the SPI inspection output reports could not be used for input to further secondary analysis. He also argued that the security inspection elements were not sufficiently decomposed into units that would allow an ambitious system administrator to formulate specific inquiries over the elements of a system's security configuration. With the SPI 3.x series, all of the major security inspection modules produce intermediate "data-share" machine-readable reports in what we call "common output report format" (CORF). A Configuration Query Language (CQL) and parser were written to provide a friendlier programming interface to the data-extraction libraries, providing support for flexible and conditional queries about

security configuration. Much of the character of the SPI 3.x series can be traced back to Gene's comments.

SPI Today

The suite of security tools provided by SPI effectively meets the problems faced by the user community, and the goals originally outlined:

Configuration Query Language (CQL) This is a high level, interpretive scripting language for extracting important system information, and presenting that information in a manner utilized by all other tools. CQL allows for complex queries to be made about files, users, and groups. Functionality that is not provided by the language may be introduced through C programming language functions.

Quick System Profile (QSP) This is an extensive CQL script, unique for each operating system, which looks for known vulnerabilities and common security problems. This tool is in a constant state of update, to address the latest known vulnerabilities. The QSP script includes tests for problems specific to a particular operating system.

Password Security Inspector (PSI) This tool is designed to uncover poorly choosen passwords. It attempts to match the user's encrypted password with common variations of the user's personal information, and words from selected custom dictionaries. An internal password inspection database is employed to allow the user to test only those passwords changed since the last time PSI was run, and allow the user to implement a discretionary password-aging policy.

Change Detector Tool (CDT) When initialized, this tool creates a database "snapshot" of important user, group, and file information. Users may define various subsets of files and accounts, and specify for each of these which attributes are suitable for change detection reporting. On subsequent executions, CDT reports changes in this information relative to the snapshot, including when files, users, and groups have been changed, added, or deleted. The system administrator may then verify that the changes made were indeed intended. CDT is used to check for unauthorized additions of user accounts, to track group memberships, and to catch changes to

file access permissions, ownerships, access and modification times, etc. File content changes may be tracked as well through the use of crypto-checksums.

Access Control Test (ACT) This is a rule-based, goal-seeking system designed to assess sequential dependencies in computer access control mechanisms. Loosely based upon Bob Baldwin's "Kuang" tool [3], this utility applies an external rulebase particular to UNIX access controls.

Binary Authentication Tool (BAT) This tool ensures that all executables and libraries that make up the operating system are up-to-date (incorporate the latest patches) and authentic (are not Trojan Horses). BAT examines host file systems to identify known vulnerable system binaries and suggests the best patch or replacement binary to correct the problem. Authentication and patch information tables are provided by the SPI development team. (In a related CSTC effort, the Binary Authentication Signatures Integrity Standard - BASIS - is working to have vendors provide and maintain authentication tables as part of their software release process.)

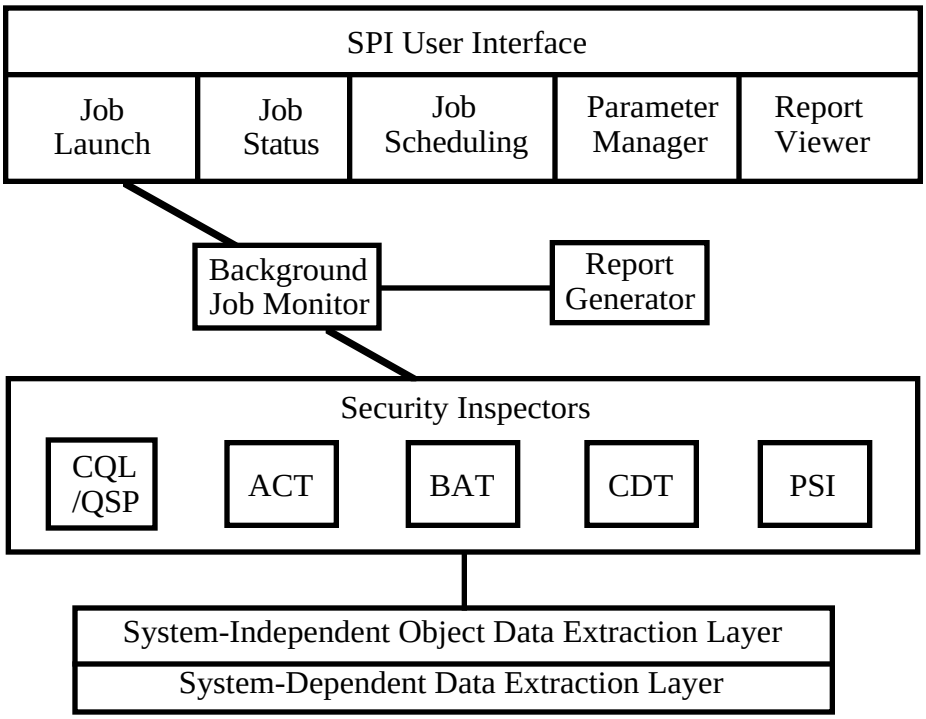
Easy Of Use

All of the SPI tools may be managed through a full-screen text-based user interface. Reasonable default values are provided for each inspection tool, and they can be scheduled to run at particular times. Every option and data field comes with on-line help specific to the current screen or data field selected, to aid novice users. For more experienced users, who wish to bypass the user interface, the command line options for all tools are fully documented through "man" pages.

All of the SPI security inspection tools produce a machine-readable Common Output Report Format (CORF). The user interface employs the SPI report generator (RG) to convert this intermediate form, producing user friendly, informative reports. Direct access to the report generator and the raw tool output is provided to the advanced SPI user. The report generator uses customizable configuration files, allowing the end user to modify existing report formats, or create new reports. By utilizing the common output format, results from multiple hosts or multiple inspection tools may be combined

and used as input to additional forms of analysis or to produce more comprehensive global reports.

The Current SPI Product Structure



Architecture of Code

SPI utilizes multiple abstraction layers, to aid portability and adaptability. At the lowest level is the operating system, which provides its own set of primitives for accessing system information. All operating system dependent code is isolated in identifiable libraries, with a well-designed object oriented interface for extracting needed information. This provides a consistent system interface for all the inspection management tools.

In this programming interface, user, group, and file objects were constructed. Each object allows for consistent access to system data, independent of the operating system. To port all tools to a new operating system, one need simply to port the object library. This design helped resolve portability issues. The recent porting of SPI to VMS demonstrated that SPI could

be ported quickly to a non-POSIX environment, largely by rewriting the system-dependent data-extraction layer.

Providing a C programming language interface to system information, however, is not sufficient to encourage end users to expand SPI's security analysis capability.

The Configuration Query Language allows for high-level queries friendlier to users. It provides easy access to system information without having to compile or understand a cryptic programming language. CQL is quite powerful, enough so that the Quick System Profile tool is actually a collection of CQL scripts. Many of the other tools utilize CQL scripts internally. Site-specific system queries can be written easily by the end user.

One important advantage of CQL is its support for C programming language extensions. If a particular check can not be done through CQL alone, a C function can be written and called from a CQL script.

Tool Design

The CORF format serves as the communication mechanism between all SPI tools. It serves as a powerful and flexible means of storing both data and results. The SPI tools utilize CQL scripts (which generate CORF output) for extracting system information. CORF also serves as the database format, storing user, group, and file information. All of the SPI inspection tools use CORF for reporting inspection results, including warnings, advisories, headings and summary information. Output from several tools may be combined together, and used to create new reports. Tools may be written which analyze output from several inspection tools, collected from a network of host machines, to produce results unobtainable from single host analysis.

Each tool is designed to be accessible by a wide range of users. Most users will feel comfortable modifying each tool's behavior through the user interface. For those who wish to have more direct access to each tool's functionality, each tool may be run from the command line. Standard UNIX 'man' pages are provided which explain command line parameters.

In short, we believe the current single-host SPI security assessment product has approached a zenith in fundamental security inspection and inspection management functionality. However, the effective use of security management personnel demands that network-wide inspections be easily conducted from a single command post, with sets of commands that automatically aggregate security inspections and report generation across large collections of host machines. This distributed inspection capability brings forth unique security management issues that the next generation SPI must address.

SPI - The Next Generation

The Management Aspects of Distributed Systems

In the shrinking domains of monolithic computing, where users far outnumber systems, one could afford the small cadre of highly skilled system administrators needed to securely manage the computing base. In contrast, the transition to distributed computing has resulted in a situation where the number of computer systems is roughly commensurate with the number of users. This has brought about a certain crisis in the secure administration of networked systems.

The individual nodes in a distributed system are not “dumb terminals” nor even single-user “personal computer” systems. They are typically multi-user, multi-tasking workstations capable of supporting scores of simultaneous user accounts, sessions, and processes. As such, they carry with them the full burden of system administration responsibilities. In the absence of effective distributed system administration, every scientist, engineer, finance or personnel manager with a networked workstation on their desk must also accept the role of system administrator, a role for which most are only marginally skilled. Expecting end-users to be the primary system administrators is both a misapplication of talent, and moreover unrealistic. This expectation is a major factor contributing to the pervasive security problems in networked systems. We need an effective means to centralize the administration of distributed systems.

In a fundamental sense, system administration is the most sensitive of information processing activities. While the

compromise of a single scientific or engineering file may jeopardize a project, the compromise of a system configuration file can jeopardize the entire computing base and all of the data files it manages. This sensitivity poses unique problems in a distributed environment.

In contrast to monolithic computing, distributed system administration requires that sensitive system configuration data pass across the network, beyond the direct control of the computing space of any particular machine. On the network, data is guided by lower level protocols providing insufficient data security, evidence the number of network-based attacks involving "packet-sniffing" [4,5] and foreign host machines masquerading [6] as trusted ones.

The long-term solution involves the revamping of these common network protocols to accommodate built-in security assurance. Until that time, such assurances must be provided in the application layer, and this is the course SPI developers have taken in the design of distributed security inspection.

SPI Distributed Security Inspections

The next-generation SPI is being designed to satisfy the following broad requirements:

1. Automated and command-driven inspection of 50 or more remote host machines from a central command host, with consideration for upward scalability.

2. Flexible inspection parameterization.

3. Flexible job scheduling and automated job control.

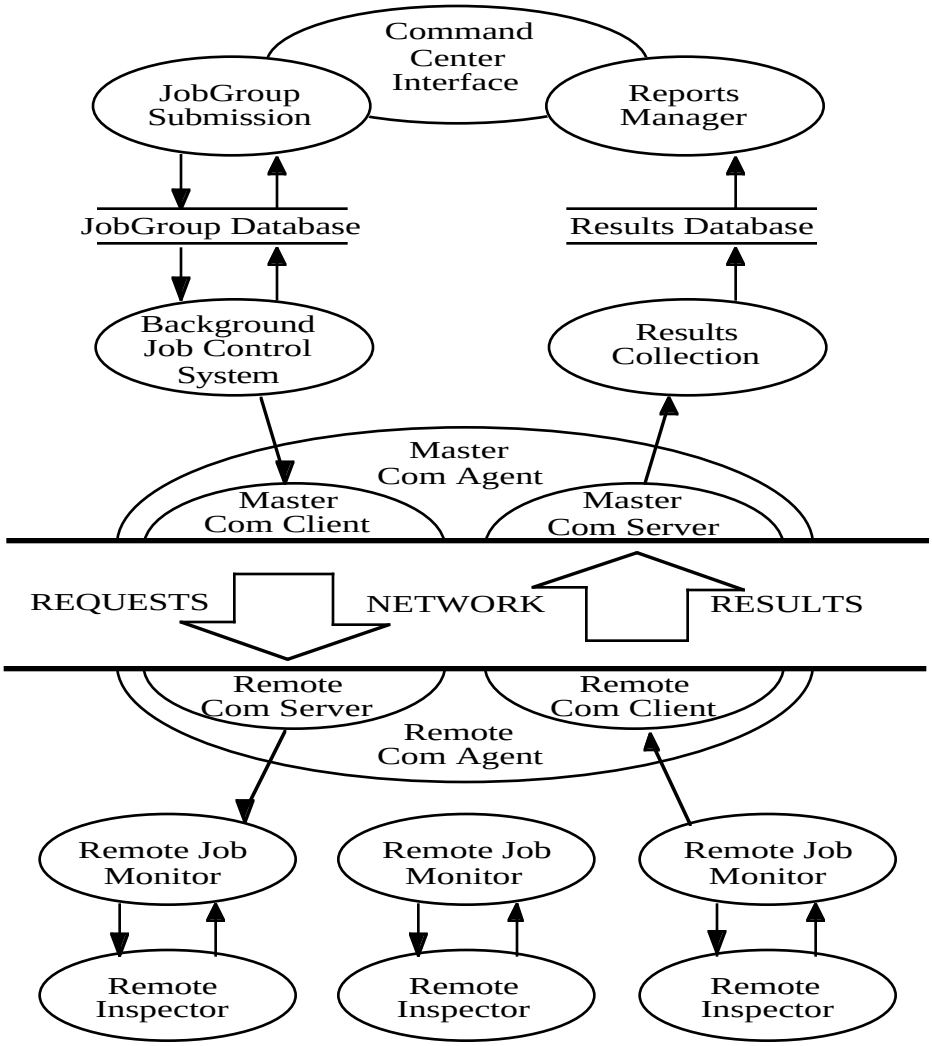
4. Flexible report aggregation capabilities.

5. Robust performance in a variable environment.

6. Host-to-host authentication, integrity assurance, and privacy assurance for all command and data traffic.

The following pages detail our plans to satisfy these distributed inspection requirements, and demonstrate how the existing layered design contributes to an effective implementation.

The SPI Distributed Security Inspection Architecture



The model we have adopted for distributed system inspections calls for a SPI command host that acts as the single "security client" collecting inspection results, with the remaining hosts of the target domain acting as "security servers." This means that the remote target hosts need only support the data extraction and analysis modules, while the user-interface and its complement of administrative functions (job scheduling, report management, online help, etc) will exist only on the SPI command host. This will go a long way to ease the portability obstacles SPI has often faced, and will even allow the integrated inspection of UNIX, VMS and other host operating systems.

The data extraction libraries and a selection of inspection modules will be installed on each of the remote hosts. For each remote host, these inspection modules will run under the direction of a security server daemon that is triggered by an authenticated signal from the command host. The remote inspections will run asynchronously, and as each inspection completes, a remote job monitor will open a connection to the SPI command host to deliver the inspection results.

The SPI Distributed Job Control System

The routine inspection of fifty or more computer systems presents logistical problems not present when dealing with systems individually. When the SPI Job Control System is tasked to perform an "AllHosts level 3 password inspection" it must launch and monitor perhaps fifty remote password inspection processes. Yet it should appear as a single operation with a single status to the security operator. Suppose that, under normal situations, it would take about one hour for each of the remote processes, occurring in parallel, to complete. How should SPI respond if:

- for the first 15 minutes, two of the target hosts repeatedly fail to respond to the job request?
- after the first 30 minutes, the SPI command host is shut down for 3 hours (or 3 days) for maintenance?
- after 90 minutes, only 46 of the 50 hosts have completed?

In each case, the entire distributed inspection system must behave in a reasonable manner. That is to say, it should respond in the same manner that a responsible human operator would if faced with similar circumstances. At any given point in time, the Job Control System may be juggling hundreds of operations; monitoring scores of remote inspection jobs, preparing to launch jobs according to predefined schedules, dealing with balky remote hosts or network connections, conferring with the report collection service to confirm successful completion of remote jobs, etc. In the midst of all this activity a power outage may shut down all systems for a time. When the power is restored, and the systems come back up, the entire operation should be able to "pick up where it left off" in as much as it would be reasonable to do so.

In order to provide such rugged behavior, it is necessary that each autonomous process maintain state by temporarily logging fundamental transactions until such time that the information is no longer critical to the overall state of the system. A great deal of consideration has been given to ensure that the distributed SPI system will provide robust consistency.

The SPI Distributed Inspection Functionality

We intend to provide the security operator the ability to:

- Define “HostGroups” designating particular subsets of host systems.

- Define “JobGroups” specifying an instance of inspection type and level for a selected HostGroup.

- Submit a JobGroup to the Job Control System, with provision for specifying a range of scheduling parameters. Examples include:

-- Schedule a job to run ONCE NOW.

-- Schedule a job to run ONCE at a given DATE and TIME.

-- Schedule a job to run PERIODICALLY, beginning at an initial DATE and TIME, and repeating every N (minutes, hours, days.)

-- For any scheduled JobGroup, allow specification of a TIMEOUT value such that jobs that either fail to launch or complete within TIMEOUT of the scheduled start time are reported and skipped.

- Query the status of scheduled JobGroups to review jobs pending, jobs active, remote hosts responding or not responding, etc.

- Specify report parameters for varied reports to be generated over the results returned by completed JobGroups.

This last point implies the ability to generate organization-wide security conformance statistics, providing support for real measures in areas ranging from the effectiveness of

security awareness activities to the penetration of malicious code during an active network attack.

Finally, by aggregating the intermediate machine-readable results from selected network-wide surveys, we will be able to provide support for future "inter-host vulnerability" analysis tools. As an example, consider that any UNIX user may place a ".rhost" entry in their home directory, allowing them to log in from any other named host-account. The wanton creation of such entries by users produces additional complexity in the analysis of resource access controls.

To make this example more tangible, say that user U1 at host M1 creates a ".rhost" entry specifying "U2 at M2". Several security questions should be asked that cannot be answered purely by inspection of host M1; Is M2 a host in our security domain? Is U2 a valid user account on M2? Are the accounts U1@M1 and U2@M2 owned by the same real user?

Short of disallowing ".rhost" entries altogether, or disabling the rlogind daemon, it would be beneficial to be able to produce "maps" of these and other inter-host relationships, to ensure that they do not violate a given access control policy.

Securing the Security Inspections

While distributed security inspection and security management can provide a single point of control in securing a domain of hosts, it can also represent a focal point for the subversion of the same security domain. Consider the implications of a remote host attack from outside the security domain masquerading as the inspection command center, and the entire consort of target hosts dutifully offering up their security vulnerabilities. Methods must be employed to ensure that the SPI command center and the hosts of the security domain have a trusted channel through which to transmit inspection commands and data.

The next-generation SPI product will employ the NIST Digital Signature Standard (DSS) to provide both host-to-host authentication and data integrity. [7,8] The command host will be fitted with a module to produce the necessary DSS certificates. For security reasons, the distribution of DSS certificates must be conducted through an out-of-band channel.

Fortunately, certificate distribution can be relatively infrequent. Several methods will be employed to protect the certificates themselves from straighforward disclosure. Where data sensitivity is especially critical, these certificates can also be employed to generate Diffie-Hellman "shared keys" to exchange discardable session keys that may in turn be used to DES-encrypt subsequent data traffic.

Summary

The past 5 years have seen SPI evolve from an "emergency collection" of three rudimentary inspection aids into a mature suite of coherently structured and integrated security inspection and management tools. The distributed SPI architecure under development will represent a new chapter in the SPI evolution. The ability to leverage the expertise of dedicated, security-saavy system administrators through application of the SPI distributed inspection system should serve to promote a comprehensive and consistent security posture across a wide and varied collection of networked systems.

Bibliography

[1] Simon Garfinkel & Eugene Spafford, Practical UNIX Security, O'Reilly & Associates, Inc. June 1991.

[2] Dan Farmer & Eugene Spafford, The COPS Security Checker System Department of Computer Sciences, Purdue University CSD-TR-993, 1990.

[3] Robert W. Baldwin, "Rule Based Analysis of Computer Security", MIT June 1987.

[4] U. S. Department of Energy - Computer Incident Advisory Capability, "Network Monitoring Attacks" CIAC Advisory Notice E-09, February 3, 1994.

[5] U. S. Department of Energy - Computer Incident Advisory Capability, "Network Monitoring Attacks Update" CIAC Advisory Notice E-12, March 18, 1994.

[6] U. S. Department of Energy - Computer Incident Advisory Capability, “Internet Address Spoofing and Hijacked Session Attacks” CIAC Advisory Notice F-08, January 23, 1995.

[7] U. S. Department of Commerce/NIST Federal Information Processing Standard, Secure Hash Standard (FIPS PUB 180-1, draft) May 31, 1994

[8] U. S. Department of Commerce/NIST Federal Information Processing Standard, Digital Signature Standard (FIPS PUB XX, draft) February 1, 1993

Technical Information Department • **Lawrence Livermore National Laboratory**
University of California • Livermore, California 94551